

Using Scripting to Increase the Impact of Highly Interactive Learning Objects

Frank Hanisch

WSI/GRIS University of Tübingen, Germany
hanisch@gris.uni-tuebingen.de

Abstract: This brief contribution describes how to employ a scripting interface for proper embedding of highly interactive learning objects (Java Applets) into Web-based courseware. We start with a short overview of our current toolkit of reusable software components that allow the development of a greater pool of virtual experiments. We illustrate how we overcome their isolation by scripting from other learning objects, e.g. from course text, illustrations, or exercises. Further we discuss how to build up a community that may create and modify scripts online by using online wizards. As proof-of-concept we demonstrate applications in the field of Scientific Visualization¹.

Introduction

Process-oriented learning environments enforce active self-learning and therefore include constructive concepts that are embodied as interactive learning objects. Today, Java applets allow for intuitive, bi-directional interaction together with high-level visualization that may be seamlessly embedded into a Web-based courseware. In 1998 we demonstrated how to reduce the immense costs that affords the development of a large pool of virtual experiments by creating a toolkit of reusable software components (Klein & Hanisch 1998). Since then, many other projects focused on developing component repositories for learning objects (Laleuf & Spalter 2001).

Unfortunately, actual profit of resulting courseware is reduced by poorly implemented interlinking of learning objects. In particular, interactive learning objects appear isolated: they neither can be modified sufficiently (e.g. by choosing parameters or enhancing functionality) nor be interlinked properly with their context (e.g. by synchronizing with a guided tour). We therefore supplied our virtual experiments with a scripting interface. Scripting enables authors and learners to steer interactive learning objects by textual commands defined within other learning objects (Christian 2000).

To enable community members to take part in the development of our courseware, we set up online wizards. Single learning objects may be modified, annotated, or rated. We deal with non-interactive *and* interactive elements, e.g. we implemented prototypes that enable authors to modify an experiment's set of scripts.

Software Components

Our component-based architecture consists of geometry objects, renderers, constraints, scene graph components, and GUI elements. Renderers encapsulate an object's visual appearance. They may hold arbitrary properties to describe their visual appearance (e.g. colors, line strength, strokes). Similar, constraints separate an object's construction (e.g. orthogonal lines) and update dependencies automatically. A plug-in architecture assures extensibility. The object itself (point, mesh, etc.) contains only primitive functionality together with a transformation cache. Adapters synchronize objects with GUI elements (e.g. a scalar with a slider) automatically. Using a hierarchical scene graph we result in default interaction behavior and visualization for all our experiments. Scene graph nodes contain a geometry object together with a matching renderer and an interaction sub-tree that provides the desired interaction behavior for our geometry object. The set of scene graph actions (rendering, picking, dragging, etc.) is extensible.

[1] <http://www.gris.uni-tuebingen.de/projects/vis>

Scripting Interface

The experiments' base component is equipped with a scripting interface. Scripts may import user-defined classes, instantiate objects, and call their methods. They also might be bound to GUI elements. Scripts are defined and documented by programmers, or, using online wizards, by community members. A well-defined state machine manages all data input steps and provides facilities for authorization, default values, undo, and preview (Figure 1). After verification of an editorial board, the given input is integrated into the script database. This entails other challenges. How to ensure bug-free and generally applicable scripts? Or, by what degree should we tolerate scripting if participants work collaboratively? Our answer is: simply do the same as for non-interactive content. Require authors to test their scripts at preview step. Evaluate and enhance scripts like any other learning object - gather access statistics, utilize a rating system, and equip scripts with forum, help, and annotation facilities.

Applications

Figure 1 illustrates how scripting may help us to overcome an interactive learning object's isolation. The current course text describes the definition and basic properties of a vector field. We embed scripts to match the experiment's vector field to the illustrations (which show a symmetric, radial, and potential field). The scripts will insert particles into our scene graph, or adapt the GUI and add text fields for a function parser. The learner may continue to work with his experiment. Other chapters contain scripts to exchange renderers for popular visualization techniques (arrow plot, colorization, streamlines, LIC).

Acknowledgements

We wish to thank our institute's member Michael Hauth for implementing the Runge-Kutta integrator, as well as our student researchers Sven Gottwald and Christian Holzer for developing our scripting interface and online wizards.

References

Klein, R., & Hanisch, F. (1998). *Using a modular construction kit for the realization of an interactive Computer Graphics course*, in Proceedings of ED-MEDIA & ED-TELECOM, AACE, USA.

Laleuf, J. R., Spalter, A. M. (2001). *Create@BROWN, A Component Repository for Learning Objects: A Progress Report*, in Proceedings of ACM JCSDL2001, Roanoke, VA.

Christian, W., & Belloni, M. (2000), *Physlets*, Prentice Hall, NJ.

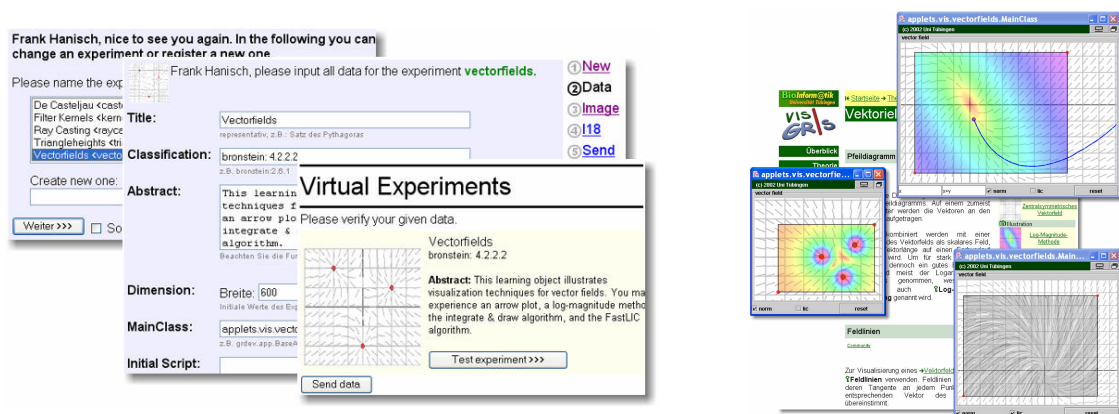


Figure 1: Online wizards for scripting virtual experiments (left side). Scripting allows for adapting an experiment's state to the current course text or illustration (right side).